# Planning vs. dynamic control: Resource allocation in corporate clouds

Andreas Wolke, Martin Bichler, Thomas Setzer

**Abstract**—Nowadays corporate data centers leverage virtualization technology to cut operational and management costs. Virtualization allows splitting and assigning physical servers to virtual machines (VM) that run particular business applications. This has led to a new stream in the capacity planning literature dealing with the problem of assigning VMs with volatile demands to physical servers in a static way such that energy costs are minimized. Live migration technology allows for dynamic resource allocation, where a controller responds to overload or underload on a server during runtime and reallocates VMs in order to maximize energy efficiency. Dynamic resource allocation is often seen as the most efficient means to allocate hardware resources in a data center. Unfortunately, there is hardly any experimental evidence for this claim. In this paper, we provide the results of an extensive experimental analysis of both capacity management approaches on a data center infrastructure. We show that with typical workloads of transactional business applications dynamic resource allocation does not increase energy efficiency over the static allocation of VMs to servers and can even come at a cost, because migrations lead to overheads and service disruptions.

**Index Terms**—capacity planning, resource allocation, IT service management

## 1 Introduction

Cloud computing has been popularized by public clouds such as Amazon's Elastic Compute Cloud[1] and nowadays several Infrastructure-as-a-Service (IaaS) providers offer computing resources on demand as virtual machines (VMs). However, due to data security and other concerns, today's businesses often do not want to outsource their entire IT infrastructure to external providers. Instead, they set up their own private or corporate clouds to manage and provide computational resources efficiently in VMs [1]. These VMs are used to host transactional business applications for accounting, marketing, supply chain management, and many other functions to internal customers where once a dedicated server was used. In this paper, we focus on corporate clouds hosting long-running transactional applications in VMs. This environment is different to public clouds, where some VMs are being deployed while others are undeployed continuously.

Server virtualization offers several advantages such as faster management and deployment of servers or the possibility to migrate VMs between different servers if required. Arguably the strongest motivation for IT service managers is increased energy efficiency through higher hardware utilization and fewer active servers. Overall, active servers are the main energy consumer in data centers besides cooling facilities. Energy usage already accounts for up to 50% or more of the total operational costs of data centers [2]. It is predicted to reach around 4.5 percent of the whole energy consumption in the USA [3]. A recent report from the *United States Environmental Protection Agency* revealed that idle servers still use 69-97% of total energy of a fully utilized server, even if all power management functions are enabled [4].

### 1.1 Static vs. Dynamic Resource Allocation

Virtualization allows for co-hosting of applications on the same physical server running a hypervisor, which ensures resource and software isolation of applications. A central managerial goal in IT service operations is to minimize the number of active virtualized servers while maintaining service quality, in particular response times. In the literature, this problem is referred to as *server consolidation* [5], [6], [7] or *workload concentration* [8]. This is a new type of *capacity planning problems*, which is different from the queuing theory models that have been used earlier for computers with a dedicated assignment of applications [9]. Server consolidation is also different from *workload scheduling* where short-term batch jobs of a particular length are assigned to servers [10]. Workload scheduling is related to classical scheduling problems, and there is a variety of established software tools such as the IBM Tivoli Workload Scheduler LoadLeveler or the open-source TORQUE Resource Manager. In contrast, workload concentration deals with the assignment of long-running VMs with seasonal workload patterns to servers. Consequently, the optimization models and resource allocation mechanisms are quite different.

Workload concentration aims for a *static (resource) allocation* of VMs to servers over time [11], [12], [13],

- *A. Wolke and M. Bichler are with the Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany.*
- *T. Setzer is with the Karlsruhe Institute of Technology, Englerstraße 14, 76131 Karlsruhe, Germany.*

*This project is supported by the Deutsche Forschungsgemeinschaft (DFG) (BI 1057/4-1).*

1. EC2, www.amazon.com/ec2/

[5]. Based on the workload patterns of VMs an allocation to servers is computed such that the total number of servers is minimized. This approach lends itself to private clouds, where there is a stable set of VMs and predictable demand patterns.[2] After the deployment of VMs on servers monitoring tools are used to detect unusual developments in the workloads and migrate VMs to other servers in exceptional cases. However, the assignment of VMs to servers is intended to be stable over a longer time horizon. At the core of these static allocation problems are high-dimensional $NP$-complete bin packing problems, and computational complexity is a considerable practical problem. Recent algorithmic advances allow solving very large problem sizes with several hundred VMs using a combination of singular-value decomposition and integer programming techniques [6].

Live migration allows moving VMs to other servers reliably during runtime. This technology is available for widely used hypervisors such as VMware's ESX [14], Xen [15], and Linux's KVM and it promises further efficiency gains. Some platforms such as VMware vSphere, or the open-source projects OpenNebula[3] and Ovirt[4] provide virtual infrastructure management and allow for the dynamic allocation of VMs to servers [16]. They closely monitor the server infrastructure to detect resource bottlenecks by thresholds. If such a bottleneck is detected or expected to occur in the future, they take actions to dissolve it by migrating VMs to different servers. Also, software vendors advocate dynamic resource allocation and provide respective software solutions for virtualized data centers [17]. We will refer to such techniques as *dynamic resource allocation* or *dynamic control*, as opposed to the *static* allocation of VMs.

Nowadays, many managers of corporate clouds consider moving to dynamic resource allocation [18] and there are various products available by commercial or open-source software providers to dynamically consolidate the VMs. Also, several academic papers on virtual infrastructure management using dynamic resource allocation illustrate high energy savings [8], [19], [20], [21], [22]. Dynamic resource allocation is less of a topic in public clouds where new VMs are being deployed and others are undeployed frequently. In such environments live migration is typically not needed, because new VMs are allocated to physical servers with low utilization, for example after VMs have been undeployed. However, when hosting long-running business applications in corporate clouds, dynamic resource allocation promises autonomic resource allocation with no manual intervention and high energy efficiency due to the possibility to respond to workload changes immediately.

For IT service managers it is important to understand if, and how much, dynamic resource allocation can save in terms of energy costs compared to static allocation. In this article, we want to address the question: *Should managers rely on dynamic resource allocation heuristics or rather use optimization-based planning for capacity management in private clouds with long-running transactional business applications?* Surprisingly, there is little research guiding managers on this question (see Section 2).

Much of the academic literature is based on simulations, where the latencies, migration overheads, and the many interdependencies of VMs, hypervisors, the network, and server hardware are difficult to model. The external validity of such simulations can be low. Therefore, experiments are important for the external validity of results. Experiments are costly, however. The set-up of a lab infrastructure including the hardware, benchmark workloads, management and monitoring software is time consuming and expensive, which might explain the lack of experimental research results to some degree.

### 1.2 Contribution and Outline

The *main contribution* of this paper is an *extensive experimental evaluation of static and dynamic resource allocation mechanisms*. More specifically, we implemented a lab infrastructure with physical servers and a comprehensive management and monitoring framework. We use benchmark business applications such as SPECjEnterprise[5] to emulate real-world business applications and model workload demand based on a large set of utilization traces from an IT service provider. Our goal is to achieve external validity of the results, but at the same time maintain the advantages of a lab environment, where the different resource allocation mechanisms can be tested and experiments can be analyzed and repeated with different workloads. Our experiments analyze different types of static and dynamic resource allocation mechanisms including pure threshold-based controllers, which are typically used in software solutions, but also ones that employ forecasting. We will use server hours used or alternatively the average number of servers used as a proxy variable to measure energy efficiency.

Our *main result* is that with typical workloads of business applications, static resource allocation leads to higher energy efficiency compared to dynamic allocation with only a modest level of overbooking. This is partly due to migration overheads and response time peaks caused by live migration. The result is robust with respect to different thresholds, even in cases where the workloads are changed significantly after the planning stage. We also implemented a simulation to cover larger scale scenarios, which uses the very same control algorithms as in the lab. We took great care to reflect system-level particularities found in the lab experiments and used parameters estimated from data in the lab.

2. Such an environment is different from public clouds, where VMs are sometimes reserved for short amounts of time for experimental purposes, or some applications exhibit very unpredictable demand as it is the case for high-traffic order entry systems that need to scale rapidly. Among the fast majority of applications run in enterprises, such applications are the exception rather than the rule.

3. opennebula.org

4. ovirt.org

5. http://www.spec.org

Interestingly, the efficiency of static allocation in larger settings increases in larger environments with several hundred VMs because the optimization can better leverage complementarities in the workloads and find more efficient allocations. The result is a clear recommendation to use optimization for capacity planning and use live migration only exceptionally.

Even though the overhead caused by live migration has been discussed [2], the impact on different resource allocation strategies has not been shown so far, but it is of high importance to IT service operations. Live migration algorithms are very efficient nowadays and the main result of our research carries over to other VM managers as we will show, because memory always needs to be transferred from one physical server to another.

## 2 Related Work

In what follows, we will revisit the literature on static and dynamic resource allocation in virtualized data centers. Note that there is substantial literature on power management in virtualized data centers including CPU frequency and voltage scaling, which we consider orthogonal to the analysis in this paper.

### 2.1 Static Resource Allocation

Research on static server consolidation assumes that the number and workload patterns of servers are known, which turns out to be a reasonable assumption for the majority of applications in most corporate data centers [6], [11], [12], [13], [5]. For example, email servers typically face high loads in the morning and after the lunch break when most employees download their emails, payroll accounting is often performed at the end of the week, while workload of a data warehouse server has a daily peak very early in the morning when managers access their reports.

The workload concentration problem is to assign VMs with seasonal workloads to servers such that the number of servers is minimized without causing server overloads. For example, Speitkamp et al. [5] show that server consolidation considering daily workload cycles can lead to 30-35% savings in servers compared to simple heuristics based on peak workloads. Mathematical optimization can be used to solve the server consolidation problem and the fundamental problem described in the above papers can be reduced to the multidimensional bin-packing problem, a known $NP$-complete optimization problem. The approach often does not scale to real-world problem sizes. A recent algorithmic approach combining singular-value decomposition and integer programming allows to solve large instances of the problem with hundreds of VMs [6]. In this paper, we will use the optimization models from Speitkamp and Bichler [5] and Setzer and Bichler [6] to determine a static allocation of VMs to servers. In contrast to earlier work, we actually deploy the resulting assignments on a physical data center infrastructure such that the approach faces all the challenges of a real-world implementation. This is a considerable extra effort beyond simulations only, but it provides evidence of the practical applicability.

### 2.2 Dynamic Resource Allocation

Live migration is nowadays available for widely used hypervisors such as VMware's ESX [14], Xen [15] as well as Linux's KVM and allows migrating a VM during runtime from one server to another. The algorithms are based on the tracking of memory write operations and memory transfers over the network that requires significant CPU and network capacity [23].

The technology allows for dynamic resource allocation without the need for planning and static assignments. All commercial and open-source approaches that we are aware of rely on some sort of threshold-based controller. It monitors the server infrastructure and is activated if certain resource thresholds are exceeded. VMs are migrated between servers in order to mitigate the threshold violation. VMware's *Distributed Resource Management* [24] and Sandpiper [25] are good examples for such systems. Gulati and Holler [24] and Ardagna et al. [21] motivate the need for workload prediction in order to avoid unnecessary migrations. In our experiments, we use both, simple threshold-based or reactive controllers and such that employ forecasting to reduce the number of back-and-forth migrations due to demand peaks.

So far there is little understanding of the benefits of dynamic resource allocation, however. According to recent surveys [18] many companies are awaiting market maturity before adopting the approach for business critical systems. A number of authors have recently proposed software frameworks for virtual infrastructure management and provide simulation results which indicate additional energy savings with dynamic resource allocation [8], [19], [20], [21], [26].

The work presented in Issarny and Schantz [26] is closely related to this paper. The authors propose *pMapper*, an energy aware VM placement and migration approach. The authors compare VM placement algorithms with respect to achievable energy savings in simulations. Some of their findings already indicate that static placement can have advantages over dynamic approaches. pMapper considers migration costs for placement decisions, and such costs are also considered in our dynamic controllers.

In contrast to prior work, we compare static and dynamic resource allocation with respect to average server demand in lab experiments using empirical data center workload traces. We argue that the external validity of lab experiments is much higher than that of simulations, and it constitutes an important complement to pure simulation studies. The simplifications of a simulation model of complex IT infrastructures always bares the risk of ignoring relevant system latencies or uncertainties in migration overheads and durations.

## 3 Experimental Infrastructure

We will now describe the hardware and software infrastructure used to conduct the experiments. First, we discuss the resource allocation mechanisms that we studied. Then, we describe the hardware infrastructure, the workloads, and the overall experimental design.

### 3.1 Resource allocation mechanisms

In our experiments we will distinguish between several types of resource allocation mechanisms: static resource allocation, reactive, and proactive control mechanisms. Static resource allocation is executed once at the beginning of an experiment. It calculates a VM to server allocation, e.g. by using a simple round robin algorithm or a mathematical program to solve the underlying optimization problem [5]. Dynamic allocation mechanisms run continuously during the experiment to reallocate VMs. Reactive controllers use utilization thresholds only, while proactive controllers employ forecasting to detect overload situations that lead to VM migrations. We implemented three types of static resource allocation mechanisms: a) Round Robin b) Optimization c) Optimization with overbooking and two types of dynamic controllers: d) Reactive and e) Proactive to be used in the experiments. These mechanisms will now be discussed in detail.

#### 3.1.1 Round robin

The round robin allocation is a simple heuristic to allocate VMs to servers a priori, before starting an experiment, and should serve as an example for a heuristic as typically used in practice. First, a number of servers is determined by adding the maximum resource demands of the VMs and then dividing by the server capacities. This is done for each resource individually and then the number of required servers is rounded to the next integer. Then the VMs are distributed in a round robin manner to the appropriate number of servers.

#### 3.1.2 Optimization and Overbooking

We used the Static Server Allocation Problem (SSAPv) [5] to compute an optimal static server allocation. We will briefly introduce the corresponding mixed integer program, which is also a basis for the algorithms used in [6].

$$
\begin{aligned}
&\min \sum_{s=1}^{S} c_s y_s \\
&s.t. \\
&\sum_{s=1}^{S} x_{sd} = 1, && \forall d \leq D \\
&\sum_{d=1}^{D} r_{dkt} x_{sd} \leq m_{sk} y_s, && \forall s \leq S, \forall k \leq K, \forall t \leq T \\
&y_s, x_{sd} \in \{0,1\}, && \forall s \leq S, \forall d \leq D
\end{aligned}
\tag{1}
$$

The program assigns $D$ VMs $d = 1, \ldots, D$ to $S$ servers $s = 1, \ldots, S$, while considering $K$ different physical server resources $k = 1, \ldots, K$ such as CPU with values between 0 and 100 for a dual-core VM. The amount of resources required by a domain (i.e., a VM) in an interval $t = [1, \ldots, T]$ is described by $r_{dkt}$ while the capacity of a server is denoted by $m_{sk}$ e.g., 200 for a quad-core server. In scenarios with overbooking, server resource capacities $m_{sk}$ are increased beyond the actual server capacity. For our experiments server capacity was overestimated by 15% (230), a value that was determined by experimentation. This accounts for the reduction of variance by adding the demand of multiple variables and leads to higher utilization with little impact on the service level, if the overbooking is at the right level.

The binary decisions variable $y_s$ indicates if a server $s$ is assigned to at least one VM and $x_{sd}$ is a binary variable that describes if VM $d$ is assigned to server $s$. With $c_s$ as costs of a server $s$, the objective function minimizes total server costs. The first set of constraints ensures that each domain is allocated to one of the servers, and the second set of constraints ensures that the aggregated resource demand of multiple domains does not exceed a server's capacity per host server, time interval, and resource type.

The optimization model was implemented using the Gurobi branch and cut solver. It requires the resource capacity of the servers as well as the workload traces as input. For the experiments, the parameters were set in accordance with the hardware specification of the data center infrastructure. Workload traces from a real-world data center were used to calculate the allocations (see Section 3.3 for more details). Various constraints are possible e.g., by covering scenarios where VMs must or must not be placed on the same server [5].

For larger problem instances, the mathematical program (1) can not be solved any more as the large number of capacity constraints and dimensions to be considered renders this task intractable. Here, we refer to a dimension as the utilization of a resource by a VM in a time interval, i.e., an unique tuple ($k$,$t$) corresponding to a particular row in the constraint matrix. Hence, a column in the constraint matrix corresponds to the workload trace of a VM for different resources. This means, the entries in a column describe a VM's utilization for $K$ server resources in $T$ time slots on $S$ servers.

Setzer and Bichler [6] describe an algorithm based on truncated singular value decomposition (SVD) which allows solving larger problems as well with near-optimal solution quality. An evaluation of the SVD-based approach using workload data from a large data center has shown that this leads to high solution quality, but at the same time allows for solving considerably larger problem instances of hundreds of VMs than what would be possible without data reduction and model transform. In our simulations, we will apply this approach to derive static server allocations with large problem sets of 90 VMs or more.

#### 3.1.3 Reactive control

The reactive controller is a dynamic mechanism aimed at migrating VMs so that the number of servers is minimized and server overload situations are counteracted. A migration is triggered if the utilization of a server exceeds or falls below a certain threshold. The controller balances the load across the servers similar to the mechanism described by Wood et al. [25]. Algorithm 1 illustrates the actions taken in each control loop.

The controller uses the Sonar[6] monitoring system to receive the CPU and memory load of all servers and VMs in a three-second interval. The data is recorded and stored in a buffer for ten minutes. Overload and underload situations are detected by a control process running every five minutes.

The function *FIND-VIOLATED-SERVERS* marks a server as overloaded or underloaded if $M = 17$ out of the last $K = 20$ CPU utilization measurements are above or below a given threshold $T_{overload}$ or $T_{underload}$. The thresholds are important as the response times depend on the utilization. An underload threshold of 40% and an overload threshold of 90% was chosen based on extensive preliminary tests described in Section 5.4.

```
Data: Servers S and VMs V
CONTROL(S, V)
    vsrv ← FIND-VIOLATED-SERVERS(S) ;
    UPDATE-VOLUME-VSR(S, V) ;
    foreach s ∈ vsrv do
        vms ← VMS-ON(s) ;
        SORT-BY-VSR(vms, DESC) ;
        for v ∈ vms do
            t ← FIND-TARGET(v, S\{s}) ;
            if t ≠ NULL then
                // Block servers for 30 seconds after
                   migration end
                MIGRATE-AND-BLOCK(s, v, t, 30) ;
                go to next s ∈ vsrv ;
            end
        end
    end
```

**Algorithm 1:** Reactive controller

Overloaded and underloaded servers are marked and handled by offloading a VM to another server. A VM on the marked server has to be chosen in conjunction with a migration target server. Target servers are chosen based on their $volume = \frac{1}{1-cpu} * \frac{1}{1-mem}$. Here, we follow a procedure introduced by Wood et al. [25]. VMs are chosen based on their $vsr = \frac{volume}{mem}$ ranking which prioritizes VMs with a high memory demand but low volume. Both, the server $volume$ and VM $vsr$ values are calculated by the function *UPDATE-VOLUME-VSR*.

The algorithm tries to migrate VMs away from the marked server in ascending order of their $vsr$. The function *VMS-ON* determines all VMs running on the source server and the function *SORT-BY-VSR* is used to sort them by $vsr$.

For each VM in this list, the algorithm described by function *FIND-TARGET* in Algorithm 2 searches through the server list to find a migration target server. For overloaded servers, migration target servers with low $volume$ are considered first, while target servers with a high $volume$ are considered first for underloaded source servers. A server is a viable migration target if the 80th percentile of the last $K$ utilization measurements for the server *lsrv* plus the ones of the VM *lvm* are lower than the overload threshold and if the target server is not blocked from previous migrations.

Only one migration is triggered at a time for each server, either an incoming or outgoing one. The migration process itself consumes resources like CPU and memory. Resource utilization readings used to decide about triggering migrations must not be not influenced by this overhead. Therefore, servers involved in a live migration are blocked for 30 seconds after the end of a live migration. The block time is described as a parameter of the *MIGRATE-AND-BLOCK* function. Subsequently they are re-evaluated for overload and underload situations. For a similar reason, the controller halts its execution during the first 2 minutes of its execution to fill its utilization measurement buffers. For the experiments the optimization algorithm described in Section 3.1.2 was used to calculate the initial allocation.

```
FIND-TARGET(v, S)
    foreach s ∈ S do
        if IS-BLOCKED(s) then
            continue ;
        end
        // Percentile over the last K measurement values
        lsrv ← PERCENTILE(s.load[-K:0], 80) ;
        lvm ← PERCENTILE(v.load[-K:0], 80) ;
        if (lsrv + lvm) < T_overhead then
            return s
        end
    end
```

**Algorithm 2:** Find target server in reactive control mechanism.

#### 3.1.4 Proactive control

The proactive controller extends the reactive one by a time series forecast to avoid unnecessary migrations. A migration will only be triggered if the forecast suggests that the overload or underload continues and is not only driven by an unforeseen spike in the demand. A forecast on time series $y_t$ is computed if a threshold violation is detected using double exponential smoothing [27] with the data forecast equation $S_t = \alpha y_t + (1-\alpha)(S_{t-1} + b_{t-1})$ and trend forecast equation $b_t = \gamma(S_t - S_{t-1}) + (1 - \gamma)b_{t-1}$ with $0 \leq \alpha, \gamma \leq 1$. Parameters were set to $b_1 = y_2 - y_1$, $\alpha = 0.2$, and $\gamma = 0.1$. We evaluated different forecasting methods such as autoregressive models (AR), using mean as forecast or simple exponential smoothing, but double exponential smoothing came out best (see Section 5.4). As the differences among several forecasting techniques on the average server demand were small, we will only report on those experiments with double exponential smoothing.

The proactive controller extends the reactive one only slightly by modifying the function *FIND-VIOLATED-SERVERS* as shown in Algorithm 3. For each server a

6. https://github.com/jacksonicson/sonar

load forecast is computed using one minute of utilization measurements. If the forecast and $M$ out of $K$ measurements pass a threshold an overload or underload situation is detected. We will see that the proactive control mechanisms cannot reduce the number of servers significantly as compared to static allocation via optimization, but they cause additional migrations. There are ways to penalize the migrations in proactive or reactive control mechanisms, but this would lead to an even higher server demand, which is why we did not consider such approaches further in this paper.

```
FIND-VIOLATED-SERVERS(S)
    for s ∈ S do
        // Forecast
        lfcst ← FORECAST(s.load) ;
        lsrv ← s.load[-K:0] ;
        // Count all server load measurements greater
           than T_overload
        ocrt ← LEN(s.load[s.load > T_overload]] ;
        ucrt ← LEN(s.load[s.load < T_underload]] ;
        s.mark = 0 ;
        if ocrt > M and lfcst > T_overload then
            s.mark = 1 ;
        end
        else if ucrt > M and lfcst < T_underload then
            s.mark = -1 ;
        end
    end
```

**Algorithm 3:** Proactive controller uses a forecast to detect violated servers

### 3.2 Hardware infrastructure

The hardware infrastructure we use to conduct the experiments consists of six identical servers and 18 VMs. Fedora Linux 16 is used as operating system with KVM as hypervisor. Hu et al. [28] show that KVM provides currently among the most efficient migration algorithms in terms of down time and migration time, which is why we have chosen it for the experiments. Each server is equipped with a single Intel Quad CPU Q9550 2.66 GHz, 16 GByte memory, a single 10,000 rpm disk and four 1GBit network interfaces. A VM is configured with two virtual CPU cores, 2 GByte memory, a single network interface and a qcow2 disk file as block storage device. All VMs and images were created prior to the experiment execution.

The VM disk files are located on two separate NFS storage servers that get mounted by all hypervisor servers. The first one is equipped with an Intel Xeon E5405 CPU, 16 GByte memory and three 1GBit network interfaces in a 802.3ad LACP bond. The second storage server is equipped with a Intel Xeon E5620 CPU, 16 GByte memory and three GBit network interfaces also in a LACP bond. Both used a RAID 10 write back configuration with enabled disk and onboard caches and a stripe size of 128 KByte. During each experiment we monitored the Linux *await*, *svctime*, and *avgqu-sz* metrics that did indicate a healthy system. We used a HP ProCurve Switch 2910al-48G with a switching capacity of 176 Gbps, sufficient to handle traffic on all ports in full-duplex operation.

A Glassfish[7] application server with the SPECjEnterprise2010[8] (SPECj) application and a MySQL database server[9] is installed on each VM. SPECj was chosen because it is widely used in industry to benchmark enterprise application servers. It is designed to generate a workload on the underlying hardware and software that is very similar to the one experienced in real world business applications.

Two additional servers are used as workload drivers. Each one is equipped with an Intel Core 2 Quad Q9400 CPU with 12 GByte main memory and two 1GBit network interfaces in an LACP bond. A modified version of the Rain[10] workload framework is used to simulate varying workload scenarios based on the three sets of workload traces MIX1-3 as described in the following section.

### 3.3 Workload

We leveraged a set of 481 raw server workload traces from a large European data center. The traces contain CPU and main memory usage in a sampling rate of five minutes over a duration of ten weeks. The servers were running enterprise applications like web servers, application servers, database servers, and ERP applications. Autocorrelation functions showed that seasonality on a daily and weekly basis is present in most of the traces as it has also been found in related papers [29] [5].

Out of all raw workload traces we sampled three distinct sets (MIX1, MIX2, MIX3) with 18 traces each. The first one comprises traces with low variance, while the second one consists of traces with high variance and many bursts. The third set is a combination of the first and second one, generated by randomly sampling nine traces from MIX1 and MIX2 without replacement.

The selected workload traces were then used to model demand for our experiments. The average resource utilization for one day was used as a demand pattern in an approach described by Speitkamp and Bichler [5]. Values of each demand pattern were normalized to a range of $[0, 1]$ by taking its maximum value over all demand patterns in a set of MIX1-3 as a reference.

Examples of MIX1 demand patterns are shown in Figure 1. Their shape does not indicate short-term bursts or random jumps. However, there can of course be an increased demand in the morning, evening, or during the operational business hours of a day compared to historical workloads. MIX2 in contrast exhibits peaks and is not as smooth as MIX1.

Each demand pattern was leveraged by a workload driver to simulate application users on a VM running the SPECj benchmark application. The number of simulated users changes according to a demand pattern that is assigned to each VM. During that process CPU

7. http://glassfish.java.net/
8. http://www.spec.org/jEnterprise2010/
9. http://www.mysql.com
10. https://github.com/yungsters/rain-workload-toolkit

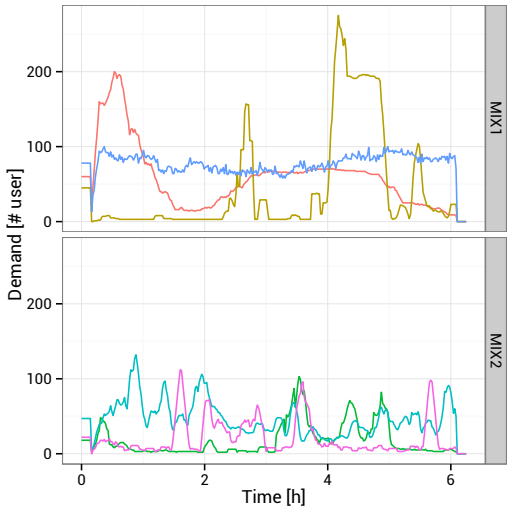


Fig. 1. Three sample workload demand traces for MIX1 and MIX2. Each describes the number of users that are simulated on a VM over 24 hours.

and memory consumption of the VM was monitored. This monitored workload trace was ultimately used to parametrized our optimization models and simulations.

All demand patterns used by our experiments are provided online [11] for reproducibility of the experiments.

The measured workload traces describe the VM utilization in values between $[0, 100]$ on each logical CPU. Each VM uses 2 CPU cores while a server has 4 CPU cores. Therefore, we will assume the capacity of one server by 200 capacity units in the default optimization scenario and 230 units in the overbooking scenario.

In addition, we conducted a second set of experiments where we took the workload mixes MIX1-3 to determine an allocation, but added noise to the workload traces which were then used to evaluate the resource allocation mechanisms. This should be a scenario, where the demand and consequently the workload traces changes significantly from those used to compute the initial allocation, and describe a challenging scenario for static resource allocation. Different patterns of noise were added to the original time series to simulate an increased demand in the morning or a reduced demand from 7 p.m. to 12 p.m.

Each modified workload trace was changed by scaling the values linearly using factors $[0.8, 1.3]$ and shifting it by $[-30, +30]$ minutes. Shifting does not alter the length of the trace. Elements which are moved beyond the traces end are re-inserted at the beginning. Table 1 describes the average difference between the default and modified workload traces for MIX1-3 versus MIX1m-3m. The table shows the difference of the mean of the original and the modified workload mix. Modified workloads contain more peak demands as shown by the 90th percentile. Spearman correlation coefficient shows

11. https://github.com/jacksonicson/times

TABLE 1
Pairwise comparison of the workload traces

| Metric | MIX1 | MIX2 | MIX3 |
|---|---|---|---|
| $mean(\bar{x_0}, .., \bar{x_n}) - mean(\bar{y_0}, .., \bar{y_n})$ | 4.29 | 5.57 | 5.58 |
| $mean(p^{50}_{x_0}, .., p^{50}_{x_n}) - mean(y_0^{p50}, .., y_n^{p50})$ | 2.60 | 5.40 | 5.05 |
| $mean(p^{90}_{x_0}, .., p^{90}_{x_n}) - mean(y_0^{p90}, .., y_n^{p90})$ | 15.61 | 13.71 | 11.68 |
| $mean(\sigma_{x_0}, .., \sigma_{x_n}) - mean(\sigma_{y_0}, .., \sigma_{y_n})$ | 6.46 | 6.01 | 4.94 |
| $mean(corr(x_0, y_0), .., corr(x_n, y_n))$ | 0.29 | 0.46 | 0.33 |

Pairwise comparison of the workload traces for MIX1-3 with the corresponding traces of MIX1-3m. All workload traces in the default mix are depicted by $x_i$ and $y_i$ is used for the modified workload traces. The 50th percentile of a time series is indicated by $p^{50}_{x_i}$.

slight similarities for MIX1 and MIX3 with their modified counterparts. There is a higher correlation for MIX2 which is mostly due to the volatile nature of the workload.

## 4 Experimental Design and Procedures

We analyze five different resource allocation mechanisms with the six workload mixes (MIX1-3 and MIX1m-3m) described in the previous section. During an experiment a number of core-metrics is recorded. The number of VMs is not varied between the experiments, nor are the threshold levels of the dynamic controllers varied. Similar to real world environments, the settings for reactive and proactive controllers are chosen based on preliminary tests with the expected workload which are described in Section 5.4.

Apart from the experiments, we also run simulations with a larger number of servers and VMs to see, if the results carry over to larger environments. We take great care to run the simulations such that the same migration overheads observed in the lab are taken into account. The detailed interactions of the SPECj application and application server are not simulated, instead the resource demands are added at a particular point in time. For this reason, we will only report the number of servers used, the number of CPU oversubscriptions, and the number of migrations. CPU oversubscriptions are calculated by counting the number of time slots where the resource demand of VMs is beyond the capacity of a server. Obviously, simulations do not have the same external validity than lab experiments, but they can give an indication of the savings to be expected in larger data centers.

In a startup phase 18 VMs are cloned from a template with Glassfish and MySQL services installed. The initial allocation is computed and the VMs are deployed on the servers according to the respective resource allocation mechanism. All VMs were rebooted to reset the operating system and clear application caches. A setup process started the Glassfish and MySQL services, loaded a database dump, configured the Rain driver with the selected user demand traces and finally triggered Rain to generate the load against the target VMs. This setup

phase is followed by a 10 minutes initialization phase during which the Rain drivers create their initial connections to Glassfish and generate a moderate workload that is equal to the first minute of the demand profile. Then the reactive or proactive controllers are started and the demand profile is replayed by Rain.

The Sonar monitoring system is used to capture relevant information in three second intervals of server and VM utilization levels. Each experiment takes six hours, where all relevant metrics such as CPU, memory, disk and network utilization are monitored. Additionally all Rain drivers reported three second averages of the response time for each service individually. This allows a complete replication of a benchmark run for analytical purposes. We report the average values of three identical runs of an experiment to account for eventually varying system latencies. Overall, the net time of experiments reported below without the initialization phase was more than 41 days.

## 5 Results

In the following we will describe the results of our experiments on the lab infrastructure as well as the results of simulations to study the behavior of the allocation mechanisms in larger environments.

### 5.1 Lab experiments with original workload mix

First we will describe the experimental results with the workloads of MIX1, MIX2, and MIX3. We will mainly report aggregated metrics such as the average and maximum response time of the services, operations per second, the number of response time violations, and the number of migrations of each six hour experiment for all VMs and applications. The values in Table 2 are averages of three runs of a six hour experiment with identical treatments. Due to system latencies there can be differences between these runs. The value in round brackets describes the variance and values in squared brackets describe the highest and lowest value. Violations state the absolute number of three-second intervals, where the response time of a request was beyond the threshold of three seconds. The service level indicates the percentage of intervals without violations.

Across all three workload sets the static allocation with overbooking had the lowest number of servers on average. This comes at the expense of higher average response times compared to other static controllers. The maximum response time is worse for reactive systems throughout. Almost all controllers achieve a service level of 99% except for proactive (MIX1) with 98.46% and overbooking (MIX2) with 97.85%.

The results of the optimization-based allocation were comparable to dynamic controllers in terms of server demand, the results of optimization with overbooking always had the lowest server demand. The average response times of the optimization-based allocation were always lower than those of the dynamic controllers. Reactive systems come at the expense of migrations, which static allocation only has in exceptional cases such as manually triggered emergency migrations. For all experiments the total number of migrations was below 36 per experiment. On average a migration is triggered every 3 hours per VM. Proactive control with time series forecasting led to a slightly lower number of servers and migrations compared to reactive control in case of MIX2 and MIX3 but triggered much more migrations for MIX1.

It is remarkable that the variance of the average response time among the three identical experimental runs increased for the reactive control strategies compared to the static ones. Even minor differences in the utilization can lead to different migration decisions and influence the results. This seems to be counteracted by proactive controllers which are more robust against random load spikes due to their time series forecasting mechanisms. We used a Welch test to compare the differences in the response times of the different controllers at a significance level of $\alpha = 0.05$. All pairwise comparisons for the different controllers and mixes were significant, except for the difference of proactive and overbooking (MIX3).

Overall, the reactive and proactive control strategies did not lead to a significantly higher efficiency compared to optimization-based static allocations. Actually, the migrations and the higher response times lead to a clear recommendation to use optimization-based static allocations with or without some level of overbooking and avoid dynamic control in these environments. A number of factors exist which can explain this result. One is the additional overhead of migrations which can also lead to additional response time violations. This overhead might compensate advantages one would expect from dynamic resource re-allocation. Some of the migrations of the reactive controller are triggered by short demand peaks and proof unnecessary afterwards. One could even imagine situations, where a controller migrates VMs back and forth between two servers as their workload bounces around the threshold levels. A proactive controller with some forecasting capabilities can filter out such demand spikes in order to avoid unnecessary migrations.

### 5.2 Lab experiments with modified workload mix

We wanted to see, if the results for the workload sets MIX1-3 carry over to a more challenging environment, where the actual demand traces during the experiment differ significantly from those used to compute a static allocation. The modified demand traces of the sets MIX1m-3m were used in the workload drivers while the static allocation was still computed with the original workload traces MIX1-3. For this reason, the average number of servers remained the same as for the first experiments for all static controllers. The results of these second experiments are described in Table 3.

One would expect that static allocations are much worse in such an environment compared to their dynamic counterparts. Interestingly, the main result carries

TABLE 2
Experimental results for various controllers

| Controller | $\overline{\text{Srv}}$ | $\overline{\text{RT}}$ | $\lceil \text{RT} \rceil$ | $\frac{\overline{\text{O}}}{[\text{sec}]}$ | $\text{O}_{\text{late}}$ | $\text{O}_{\text{fail}}$ | Mig | SQ |
|---|---|---|---|---|---|---|---|---|
| **MIX 1** | | | | | | | | |
| Round Robin | 6 (0) | 352 | 20186 (972) | 151 (0) | 138 (6) | 12 (2) | 0 [0/0] | 99.89% |
| Optimization | 6 (0) | 330 | 17621 (3777) | 151 (0) | 137 (26) | 8 (2) | 0 [0/0] | 99.89% |
| Overbooking | 5 (0) | 466 | 19103 (2811) | 149 (0) | 647 (181) | 10 (4) | 0 [0/0] | 99.5% |
| Proactive | 5.95 (0.07) | 566 | 42012 (5958) | 147 (2) | 1990 (1609) | 15 (5) | 10.33 [9/12] | 98.46% |
| Reactive | 6 (0) | 392 | 21501 (9386) | 150 (1) | 279 (25) | 14 (1) | 0.33 [0/1] | 99.78% |
| **MIX 2** | | | | | | | | |
| Round Robin | 6 (0) | 388 | 17016 (15823) | 81 (0) | 289 (11) | 5 (1) | 0 [0/0] | 99.78% |
| Optimization | 4 (0) | 467 | 16875 (7071) | 80 (1) | 637 (156) | 6 (4) | 0 [0/0] | 99.51% |
| Overbooking | 3 (0) | 744 | 34498 (7538) | 77 (0) | 2783 (106) | 5 (3) | 0 [0/0] | 97.85% |
| Proactive | 3.93 (0.2) | 535 | 65337 (22243) | 79 (0) | 777 (184) | 22 (17) | 23 [16/34] | 99.4% |
| Reactive | 4.34 (0.18) | 547 | 71153 (23498) | 79 (1) | 842 (359) | 28 (23) | 26.4 [18/36] | 99.35% |
| **MIX 3** | | | | | | | | |
| Round Robin | 6 (0) | 377 | 12590 (2698) | 107 (0) | 111 (13) | 8 (2) | 0 [0/0] | 99.91% |
| Optimization | 5 (0) | 347 | 11222 (1171) | 107 (0) | 73 (6) | 8 (2) | 0 [0/0] | 99.94% |
| Overbooking | 4 (0) | 483 | 21387 (1515) | 106 (0) | 673 (143) | 8 (2) | 0 [0/0] | 99.48% |
| Proactive | 4.76 (0.16) | 475 | 54636 (215) | 106 (0) | 545 (93) | 12 (4) | 14.33 [10/17] | 99.58% |
| Reactive | 4.85 (0.12) | 505 | 59651 (9129) | 105 (1) | 635 (158) | 19 (11) | 17 [17/17] | 99.51% |

Experimental results on static vs. dynamic VM allocation controllers. $\overline{\text{Srv}}$ – average server demand, $\overline{\text{RT}}$ $[ms]$ – average response time, $\lceil \text{RT} \rceil$ $[ms]$ – maximum response time, $\frac{\overline{\text{O}}}{[\text{sec}]}$ – average operations per second, $\text{O}_{\text{late}}$ – late operations count, $\text{O}_{\text{fail}}$ – failed operations count, Mig – VM migration count, SQ [%] – service quality based on 3 second intervals

over. The service levels were high and average response times were low in all treatments. Again, we used a Welch test to compare the differences in the response times of the different controllers at a significance level of $\alpha = 0.05$. All pairwise comparisons for the different controllers and mixes were significant, except for overbooking to proactive in MIX1m ($p = 0.01$), reactive to proactive in MIX2m ($p = 0.87$), and optimization to reactive in MIX3M ($p = 0.14$). For overbooking in MIX2m and MIX3m an increased average and maximum response time with a service level degradation to 91.74% and 96.37% was observed. Dynamic controllers showed a service level degradation for MIX1m with 96.81% for the reactive and 97.74% for the proactive controller. This can be explained by the overall workload demand, which is close to what the six servers were able to handle. The average server utilization was 80% over the complete six hours and all servers. As a result average response times have increased for all controllers compared to the first experiments. In this case even slightly suboptimal allocations result in a degradation of service quality during periods of high utilization which especially affects the overbooking and dynamic controllers. The optimization-based allocation in contrast still has a good service quality above 99% with fewer servers and comparably low average response times.

Comparing the throughput in operations per second with the first experiments shows an increase for MIX2m-3m. For MIX1m no increase could be found despite the fact that the demand trace of MIX1m are increased compared to MIX1 (see Table 1). Again, this is caused by the server overload situations in the MIX1m scenario.

For MIX2m-3m the dynamic controllers showed a similar behavior as for MIX2-3. The average response time remained constant while the max. response times were again increased compared to static controllers. Interestingly, the controllers were able to maintain a service quality above 99% by an increased average server count.

For all workloads the dynamic controllers triggered the same number or more migrations compared to the first experiments. However, for MIX2m, the volatile workload scenario, the migration counter of the reactive controller was substantially increased with 45 migrations on average while the proactive controller required only 20.5 migrations. Again, the variance in the average response time tends to be higher for dynamic controllers. Overall, even in scenarios where workload volatility increases for all VMs, the static optimization-based allocations perform still well.

Another working paper of our group describes a set of initial experiments on the same hardware, but with an entirely different software infrastructure with a different hypervisor (Citrix XenServer), a different threshold for the reactive controller, different operating systems, and a different workload generator [30]. While the infrastructure was less stable and focused on the evaluation of reactive control parameters, also these initial experiments found that the static allocation and a modest level of overbooking yielded low energy costs and higher response times compared to reactive control. These initial experiments used $T_{Underload}$ thresholds of 20% and 30% and $T_{Overload}$ thresholds of 75% and 85% for the reactive controller. However, efficient thresholds depend on the workload and is certainly not an easy task for IT service managers. Overall, this provides some evidence that our main result carries over to different implementations of the reactive controller, the thresholds used, hypervisors, or different samples of the workload.

Note that the results hold during business hours of a day or for data centers with customers in different time-zones. In regional data centers, where all business

TABLE 3
Experimental results for modified workload mixes MIX1m-3m.

| Controller | $\overline{\text{Srv}}$ | $\overline{\text{RT}}$ | $\lceil \text{RT} \rceil$ | $\frac{\overline{\text{O}}}{[\text{sec}]}$ | $\text{O}_{\text{late}}$ | $\text{O}_{\text{fail}}$ | Mig | SQ |
|---|---|---|---|---|---|---|---|---|
| **MIX 1m** | | | | | | | | |
| Round Robin | 6 (0) | 449 | 25087 (2911) | 165 (0) | 829 (105) | 10 (3) | 0 [0/0] | 99.36% |
| Optimization | 6 (0) | 440 | 27080 (6945) | 165 (0) | 983 (81) | 11 (5) | 0 [0/0] | 99.24% |
| Overbooking | 5 (0) | 618 | 27247 (4516) | 160 (3) | 2012 (369) | 49708 (86071) | 0 [0/0] | 98.45% |
| Proactive | 5.96 (0.07) | 600 | 55370 (28537) | 162 (2) | 2928 (828) | 858 (1692) | 7.75 [4/13] | 97.74% |
| Reactive | 5.99 (0) | 710 | 47025 (17355) | 160 (0) | 4133 (787) | 20 (3) | 14.33 [12/18] | 96.81% |
| **MIX 2m** | | | | | | | | |
| Round Robin | 6 (0) | 375 | 13717 (5646) | 101 (0) | 368 (38) | 5 (1) | 0 [0/0] | 99.72% |
| Optimization | 4 (0) | 441 | 20766 (8736) | 101 (0) | 366 (32) | 6 (1) | 0 [0/0] | 99.72% |
| Overbooking | 3 (0) | 1401 | 60584 (11310) | 90 (0) | 10699 (128) | 64 (95) | 0 [0/0] | 91.74% |
| Proactive | 4.79 (0.03) | 511 | 82047 (26877) | 99 (0) | 807 (127) | 31 (21) | 22 [20/25] | 99.38% |
| Reactive | 4.94 (0.05) | 545 | 78349 (15210) | 98 (1) | 1095 (264) | 338 (586) | 45 [40/50] | 99.16% |
| **MIX 3m** | | | | | | | | |
| Round Robin | 6 (0) | 382 | 18623 (4912) | 128 (0) | 179 (34) | 10 (2) | 0 [0/0] | 99.86% |
| Optimization | 5 (0) | 486 | 25757 (3737) | 127 (0) | 1290 (83) | 11 (2) | 0 [0/0] | 99% |
| Overbooking | 4 (0) | 823 | 31582 (1571) | 123 (0) | 4706 (200) | 11 (2) | 0 [0/0] | 96.37% |
| Proactive | 5.5 (0.16) | 465 | 49300 (13668) | 127 (1) | 802 (415) | 19 (5) | 18 [12/24] | 99.38% |
| Reactive | 5.6 (0.03) | 485 | 74017 (16339) | 126 (1) | 774 (317) | 27 (18) | 23.67 [18/33] | 99.4% |

Experimental results for mixes MIX1m-3m. $\overline{\text{Srv}}$ – average server demand, $\overline{\text{RT}}$ $[ms]$ – average response time, $\lceil \text{RT} \rceil$ $[ms]$ – maximum response time, $\frac{\overline{\text{O}}}{[\text{sec}]}$ – average operations per second, $\text{O}_{\text{late}}$ – late operations count, $\text{O}_{\text{fail}}$ – failed operations count, Mig – VM migration count, SQ [%] – service quality based on 3 second intervals

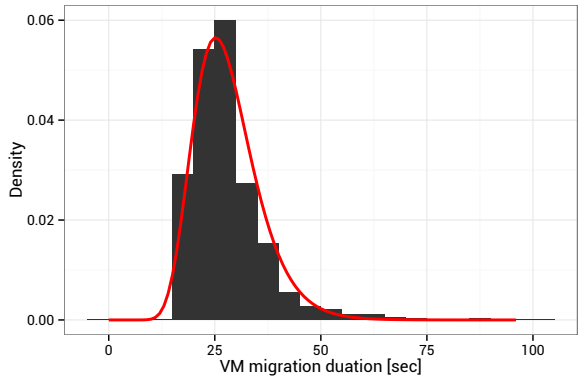


Fig. 2. Histogram of the observed live migration time.

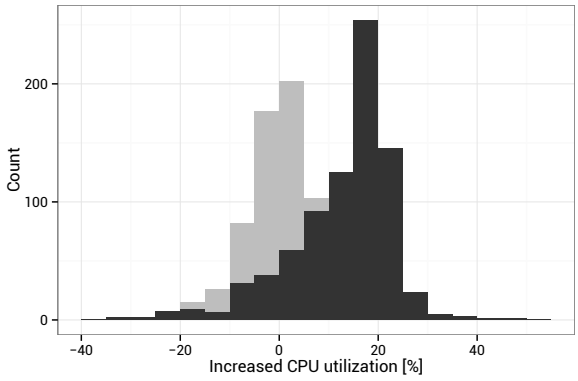


Fig. 3. Live migration CPU overhead on the source-server. All (gray) and servers with $\leq 85\%$ load (black).

applications exhibit a very low utilization at night time, it can obviously save additional energy to consolidate the machines after working hours. Such nightly workload concentrations can be triggered automatically and in addition to the static allocation.

### 5.3 Migration Overheads

During our experiments almost 1500 VM migrations were triggered. Here, we want to briefly discuss the resource overhead by live migrations, in order to better understand the results of the experiments described in the previous subsections. The mean live migration duration was 28.73s for 1459 migrations with quartiles 17.98s, 24.03s, 31.77s, and 96.73s. It follows a log-normal distribution with $\mu = 3.31$ and $\sigma = 0.27$ as shown in Figure 2.

Live migration algorithms work by tracking the write operations on memory pages of a VM which consumes additional CPU cycles in the hypervisor [23]. Both, dynamic and reactive controllers triggered only one migration at a time for each server. For each migration the mean CPU load for 60s before the migration and during the migration was calculated. Both values were subtracted which provides an estimate for the CPU overhead of a migration.

On the source server an increased CPU load with a mean of 7.88% and median of 8.06% was observed. Not all deltas were positive as seen in Figure 3 which can be explained by the varying resource demand during the migration on other VMs running on the same server. Only servers with a CPU utilization below 85% were considered for the histogram. The gray histogram area considers all migrations. In this case, many migrations did not lead to a CPU overhead as utilization cannot increased beyond 100%. For the target servers the CPU utilization increased by 12.44% on average.

Network utilization is one of the main concerns when using migrations. Similar to today's data centers, all network traffic was handled by a single network in-

terface. Similar to CPU, we calculated the delta of the network throughput before and during migrations. The difference on the source and target server was close to 70 MByte/s. Benchmarks report a maximum throughput of 110 MByte/s for a 1 GBit/s connection. This throughput is not achieved as our measurements include some seconds before and after the network transfer phase of a migration. Also, a migration is not a perfect RAM to RAM transfer as the algorithm has to decide on which memory pages to transfer. The 95th percentile of our network throughput measurements during a migration was 105 MByte/s which is close to the throughput reported in benchmarks. Network overloads were ruled out due to the use of a LACP bond with two 1 GBit/s connections where one was dedicated for live migrations.

### 5.4 Sensitivity analysis

As in any experiment, there is a number of parameter settings which could further impact the result. Especially, for reactive and proactive control approaches parameters such as the threshold levels for migrations were chosen based on preliminary tests. In the following, we want to provide sensitivity analysis in order to understand the robustness of the results. We conducted experiments varying the parameters: $T_{underload}$, $T_{Overload}$, $K$ and $M$ variables for the dynamic controllers, as well as the control loop interval. MIX2 was chosen because it entails a high variability and it is better suited to dynamic controllers. The results are described in Table 4.

Changing the threshold settings from $T_{underload} = 40$ and $T_{Overload} = 90$ to an $T_{underload} = 20$ only results in a less aggressive controller with a better performance regarding migrations, violations and average response time. This comes at the cost of an increased average server consumption. Setting $T_{underload} = 60$ made the controller more aggressive. Average server consumption could be minimized at the price of increased response time, migrations and violations. The threshold settings certainly depends on the type of workload used. They need to be tuned for each situation and to the service level requirements. For the experiments we chose a middle way considering migrations and violations.

We decreased the control loop interval from 300 to 30 seconds with negligible impact on the metrics. The average number of servers, violations and response time are comparable to the results that we found in previous experiments while the number of migrations was slightly increased.

The $K$ and $M$ values describe for how long an overload situations has to last until a controller acts upon it. Setting $K = 50, M = 45$ had a slightly positive effect on all metrics except for the migration count. Changing it to $K = 10, M = 8$ yielded a more aggressive controller with more migrations. It triggered 60 migrations compared to 26.4 before without a positive effect on average server consumption, which actually was increased.

In addition we tested different forecast settings for the proactive controller. We used an auto-regressive model (AR) instead of a double exponential smoothing. The average server count, migration count, average response time, and max. resp. time were on the same level as for previous experiments. Setting $M = \infty$ calculates the AR forecast with all available utilization readings in the controller. The average server count did not change but a negative effect on the violation count was found.

Modifying the $\alpha = 0.2$ and $\gamma = 0.1$ variables of the double exponential smoothing (DES) had no significant effect either. Increasing $\alpha = 0.5$ yielded similar results then the default configuration. $\gamma = 0.3$ resulted in more violations and slightly increased average response times without an effect on the average server count.

### 5.5 Simulations

We first wanted to understand how the results of our simulations compare to those of lab experiments, considering the parameter settings and migration overheads learned in the lab. In case simulations will yield comparable results, we want to understand, how the performance metrics develop with growing environments regarding more servers and VMs.

Our discrete event simulation framework consists of a workload driver, a controller and a model. Servers and VMs are stored in the model together with their allocation. A unique workload trace is assigned to each VM. The driver iterates over all VMs and updates their current CPU load according to their workload trace in three second intervals – the same frequency as utilization measurements are received from Sonar during an experimental run.

The framework does not reproduce the detailed interactions of web, application, and database server in a VM. It sums the workloads of all VMs to estimate the utilization of a server at a point in time. Therefore, we do not report response times or operations per second. Instead we count the time slots with CPU overload, i.e., where the accumulated CPU load of a server exceeds its capacity.

The controller is activated every five minutes and triggers migrations according to the server load status. The same model and controller implementations are used as in the experiments. Migration time is simulated using a log-normal distribution with the parameters we experienced during the experiments (described in Section 5.3). Additionally, a CPU overhead of 8% on the source and 13% on the targed server was simulated during migrations. For the simulations with MIX1-3 the same utilization traces as for the experiments were used.

Table 5 shows the results of simulations with six servers and 18 VMs to see if the results of the simulation are comparable to those of the lab experiments. The results reveal that the allocation of VMs to servers and, hence, the total amount of allocated servers in the simulations equals the amount computed in the experiments for the same scenario when using static allocation. However, this does not hold for the number of servers needed

TABLE 4
Experiments to test the sensitivity of reactive and proactive controller parameters.

| Controller | $\overline{\text{Srv}}$ | $\overline{\text{RT}}$ | $\lceil \text{RT} \rceil$ | $\frac{\overline{\text{O}}}{[\text{sec}]}$ | $\text{O}_{\text{late}}$ | $\text{O}_{\text{fail}}$ | Mig | SQ |
|---|---|---|---|---|---|---|---|---|
| **MIX2 + Reactive Controller** | | | | | | | | |
| $K = 10, M = 8$ | 4.42 (0.03) | 564 | 75110 (14560) | 79 (0) | 930 (239) | 37 (24) | 40.67 [24/60] | 99.28% |
| $K = 50, M = 45$ | 4.03 (0.11) | 536 | 71797 (15188) | 79 (0) | 771 (367) | 30 (12) | 21.33 [16/29] | 99.41% |
| $T_{Underload} = 20$ | 5.57 (0.73) | 502 | 49194 (28580) | 80 (0) | 747 (382) | 12 (11) | 11.33 [9/15] | 99.42% |
| $T_{Underload} = 60$ | 3.96 (0.1) | 571 | 81481 (26900) | 79 (0) | 1001 (143) | 46 (17) | 43.33 [28/63] | 99.23% |
| control interval = 30 | 4.22 (0.06) | 584 | 72763 (14529) | 78 (1) | 803 (313) | 47 (1) | 39.33 [31/48] | 99.38% |
| **MIX2 + Proactive Controller** | | | | | | | | |
| AR forecast | 4.15 (0.25) | 539 | 56599 (2138) | 79 (0) | 755 (155) | 22 (7) | 23.33 [20/29] | 99.42% |
| AR forecast $M = \inf$ | 3.78 (0.5) | 641 | 58231 (22600) | 78 (1) | 1671 (1182) | 12 (8) | 19.33 [7/29] | 98.71% |
| DES $\alpha = 0.2, \gamma = 0.3$ | 3.91 (0.65) | 650 | 57861 (21786) | 78 (1) | 1516 (1254) | 25 (29) | 22.33 [7/37] | 98.83% |
| DES $\alpha = 0.5, \gamma = 0.1$ | 3.85 (0.08) | 533 | 72645 (30836) | 78 (3) | 674 (136) | 37108 (64229) | 21.33 [19/24] | 99.48% |

AR forecast = Autoregressive Model, DES = Double Exponential Smoothing. $\overline{\text{Srv}}$ – average server demand, $\overline{\text{RT}}$ $[ms]$ – average response time, $\lceil \text{RT} \rceil$ $[ms]$ – maximum response time, $\frac{\overline{\text{O}}}{[\text{sec}]}$ – average operations per second, $\text{O}_{\text{late}}$ – late operations count, $\text{O}_{\text{fail}}$ – failed operations count, Mig – VM migration count, SQ $[\%]$ – service quality based on 3 second intervals

TABLE 5
Simulations for MIX1-3

| Controller | $\overline{\text{Srv}}$ | $\lfloor \text{Srv} \rfloor$ | $\lceil \text{Srv} \rceil$ | Mig | SQ |
|---|---|---|---|---|---|
| **MIX 1** | | | | | |
| Optimization | 6.00 | 6.00 | 6.00 | 0.00 | 100.00 |
| Overbooking | 5.00 | 5.00 | 5.00 | 0.00 | 84.71 |
| Proactive | 5.62 | 4.00 | 6.00 | 30.00 | 99.16 |
| Reactive | 5.74 | 5.00 | 6.00 | 34.00 | 98.99 |
| RoundRobin | 6.00 | 6.00 | 6.00 | 0.00 | 96.88 |
| **MIX 2** | | | | | |
| Optimization | 4.00 | 4.00 | 4.00 | 0.00 | 100.00 |
| Overbooking | 3.00 | 3.00 | 3.00 | 0.00 | 90.56 |
| Proactive | 3.75 | 3.00 | 6.00 | 40.00 | 98.23 |
| Reactive | 4.22 | 3.00 | 5.00 | 33.00 | 98.63 |
| RoundRobin | 6.00 | 6.00 | 6.00 | 0.00 | 100.00 |
| **MIX 3** | | | | | |
| Optimization | 5.00 | 5.00 | 5.00 | 0.00 | 100.00 |
| Overbooking | 4.00 | 4.00 | 4.00 | 0.00 | 95.31 |
| Proactive | 4.69 | 3.00 | 6.00 | 43.00 | 99.21 |
| Reactive | 5.02 | 4.00 | 6.00 | 35.00 | 98.64 |
| RoundRobin | 6.00 | 6.00 | 6.00 | 0.00 | 98.77 |

$\overline{\text{Srv}}$ – average server demand, $\lceil \text{SD} \rceil$ – maximum server demand, $\lfloor \text{SD} \rfloor$ – minimum server demand, Mig – VM migration count, SQ $[\%]$ – service quality based on 3 second intervals

by dynamic controllers although the average number of servers closely matches the experimental results.

In terms of service quality, simulations could predict the efficiency of allocation mechanisms that we observed in experiments well. This is also due to the fact that we could parametrize the simulations with values observed in the lab. In contrast, simulations usually overestimated the number of migrations triggered during experiments. The reason for this is that in the lab the OS schedules the requests, which results in a smoothing of workloads over time. In the simulation the loads of different VMs are added, leading to different migration decisions.

The comparison between simulation and experiment show that simulation results need to be interpreted with care, even if the same software infrastructure and parameter estimates are used. While there are differences in the number of servers used, the differences are small. Hence, we use simulations as an estimator to assess how the average server consumption will develop in larger environments.

We examined scenarios up to 360 VMs and approximately 60 servers. As MIX1-3 only contain 18 utilization traces each, new workload traces for the simulation are generated from the set of 481 raw workload traces. These traces were prepared as described in Section 3.3 and described as MIXSIM. The simulation results for environments with 18, 90, 180, and 360 VMs are shown in Table 6. For each treatment three simulations are conducted and their mean value is reported. Each time the set of workload traces assigned to the VMs is sampled randomly from MIXSIM.

For the static server allocation problem, computational complexity increases with the number of servers and VMs. Optimizations with six servers are still solvable with traces at a sampling rate of three minutes while problem instances with 30 or more servers are only solvable at a sampling rate of 1 hour without an optimal solution within 60 minutes calculation time, which leads to decreasing solution quality. The computational complexity and the empirical hardness of the problem was discussed by [5]. Hence, for larger problem sizes of 60 VMs or more, we computed allocations based on the algorithms introduced by Setzer and Bichler [6]. They leverage singular-value decomposition and compute near-optimal solutions even for larger problem sizes with several hundred VMs.

Figure 4 shows that with an increased number of VMs the number of servers required increases in all controllers, but that the gradient of the optimization-based controllers is much lower. Consequently, the advantage of optimization-based static allocation actually increase with larger numbers of VMs.

## 6 Conclusion

Dynamic resource allocation is often seen as the next step of capacity management in data centers promising high efficiency in terms of average servers demand. Unfortunately, there is hardly any empirical evidence for the benefits of dynamic resource allocation so far. In this paper, we provide the results of an extensive experimental study on a real data center infrastructure.

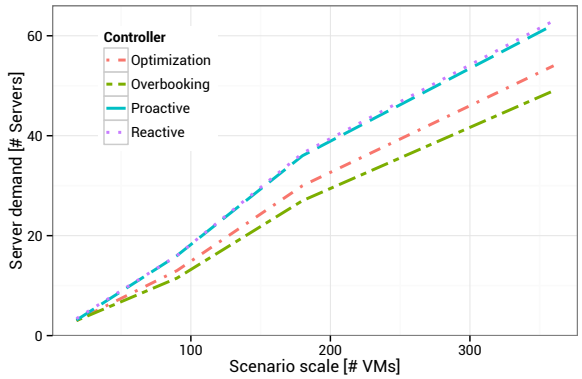


Fig. 4. Growth of the number of servers required for different numbers of VMs.

TABLE 6
Simulations for MIXSIM

| Controller | $\overline{\text{Srv}}$ | $\lfloor \text{Srv} \rfloor$ | $\lceil \text{Srv} \rceil$ | Mig | SQ |
|---|---|---|---|---|---|
| **Tiny (18 VMs)** | | | | | |
| Optimization | 3.00 | 3.00 | 3.00 | 0.00 | 100.00 |
| Overbooking | 3.00 | 3.00 | 3.00 | 0.00 | 92.17 |
| Proactive | 3.06 | 3.00 | 3.10 | 1.90 | 99.93 |
| Reactive | 3.12 | 3.00 | 3.40 | 2.80 | 99.85 |
| **Small (90 VMs)** | | | | | |
| Optimization | 13.00 | 13.00 | 13.00 | 0.00 | 98.84 |
| Overbooking | 11.50 | 11.50 | 11.50 | 0.00 | 86.88 |
| Proactive | 15.04 | 14.50 | 16.00 | 33.50 | 99.52 |
| Reactive | 15.13 | 14.50 | 16.00 | 41.50 | 99.57 |
| **Medium (180 VMs)** | | | | | |
| Optimization | 30.00 | 30.00 | 30.00 | 0.00 | 99.86 |
| Overbooking | 27.00 | 27.00 | 27.00 | 0.00 | 96.99 |
| Proactive | 32.48 | 30.00 | 36.00 | 39.00 | 99.74 |
| Reactive | 32.65 | 29.50 | 36.50 | 62.50 | 99.64 |
| **Large (360 VMs)** | | | | | |
| Optimization | 54.00 | 54.00 | 54.00 | 0.00 | 98.91 |
| Overbooking | 49.00 | 49.00 | 49.00 | 0.00 | 87.19 |
| Proactive | 59.58 | 57.00 | 62.20 | 82.20 | 99.79 |
| Reactive | 59.80 | 57.00 | 63.00 | 122.60 | 99.72 |

$\overline{\text{Srv}}$ – average server demand, $\lceil \text{SD} \rceil$ – maximum server demand, $\lfloor \text{SD} \rfloor$ – minimum server demand, Mig – VM migration count, SQ [%] – service quality based on 3 second intervals

We focus on private cloud environments with a stable set of business applications that need to be hosted as VMs on a set of servers. We leverage data from a large IT service provider to generate realistic workloads, and find that reactive or proactive control mechanisms do not decrease average server demand. Depending on the configuration and the threshold levels chosen they can lead to a large number of migrations, which negatively impact the response times and can even lead to network congestion in larger scenarios. Simulations showed that optimization-based static resource allocation provides even better results compared to dynamic controllers for large environments as possibilities to leverage workload complementarities in the optimization increase with the number of VMs.

Any experimental study has limitations and so has this. First, a main assumption of the results in this paper is the workload, which is characterized in the supplemental material. We have analyzed workloads with high volatility and even added additional noise in the demand, and the results were robust. The results do not carry over to applications that are difficult to forecast. For example, order entry systems can experience demand peaks based on marketing campaigns, which are hard to predict from historical workloads. Also, sometimes VMs are set up for testing purposes and are only needed for a short period of time. In such cases, different control strategies are required and reactive control clearly has its benefits in such environments. Such applications are typically hosted on in a separate cluster, and we leave the analysis of such workloads for future research. Second, the experimental infrastructure was small and the results for larger environments with 120 and more VMs are based on simulation. While simulation has its limitations, we took great care that the main system characteristics such as migration duration were appropriately modeled. Also, the controller software was exactly the same as the one used in the lab experiments. Finally, one can think of alternative ways to implement the reactive and proactive controllers. For example, advanced workload prediction techniques could be used [31], [32]. We conjecture, however, that the basic trade-off between migration costs and efficiency gains by dynamic resource allocation will persist also with smarter control strategies with similar workloads.

Although the study shows that with a stable set of business applications static resource allocation with a modest level overbooking would lead to the lowest average server demand, we suggest that in everyday operations, a combination of both mechanisms where allocations are computed for a longer period of time and exceptional workload peaks are treated by a dynamic control mechanism should be put in place. We argue, however, that such live migrations should be used in exceptional instances only and capacity planning via optimization should be used as an initial means to allocate VMs to servers, in environments with long-running and predictable application workloads.

## References

[1] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, Sep. 2009. [Online]. Available: http://dx.doi.org/10.1109/MIC.2009.119

[2] D. Filani, J. He, S. Gao, M. Rajappa, A. Kumar, P. Shah, and R. Nagappan, "Dynamic Data Center Power Management: Trends, Issues, and Solutions," *Intel Technology Journal*, February 2008.

[3] W.-c. Feng, X. Feng, and R. Ge, "Green Supercomputing Comes of Age," *IT Professional*, vol. 10, no. 1, pp. 17–23, Jan. 2008.

[4] V. Radulovic, "Recommendations for Tier I ENERGY STAR Computer Specification," United States Environmental Protection Agency, Pittsburgh, PA, Tech. Rep., October 2011.

[5] B. Speitkamp and M. Bichler, "A Mathematical Programming Approach for Server Consolidation Problems in Virtualized Data Centers," *IEEE Transactions on Services Computing*, vol. 3, no. 4, pp. 266–278, 2010.

[6] T. Setzer and M. Bichler, "Using Matrix Approximation for High-Dimensional Discrete Optimization Problems: Server Consolidation Based on Cyclic Time-Series Data," *European Journal of Operational Research*, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.ejor.2012.12.005

[7] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 215–228, 2013.

[8] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, "EnaCloud: An Energy-Saving Application Live Placement Approach for Cloud Computing Environments," *Cloud Computing, IEEE International Conference on*, pp. 17–24, 2009.

[9] J. H. Son and M. H. Kim, "An analysis of the optimal number of servers in distributed client/server environments," *Decision Support Systems*, vol. 36, no. 3, pp. 297–312, Jan. 2004. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167923602001422

[10] C. Bodenstein, G. Schryen, and D. Neumann, "Energy-aware workload management models for operation cost reduction in data centers," *European Journal of Operational Research*, vol. 222, no. 1, pp. 157–167, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221712002814

[11] K. Parent, "Consolidation Improves IT's Capacity Utilization," Court Square Data Group, Tech. Rep., 2005.

[12] J. Rolia, L. Cherkasova, M. Arlitt, and A. Andrzejak, "A capacity management service for resource pools," in *Proceedings of the 5th international workshop on Software and performance*, ser. WOSP '05. New York, NY, USA: ACM, 2005, pp. 229–237. [Online]. Available: http://doi.acm.org/10.1145/1071021.1071047

[13] T. Setzer, M. Bichler, and B. Speitkamp, "Capacity Management for Virtualized Servers," in *INFORMS Workshop on Information Technologies and Systems (WITS)*, Milwaukee, USA, 2006.

[14] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 25–25. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247360.1247385

[15] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, ser. SOSP '03. New York, NY, USA: ACM, 2003, pp. 164–177. [Online]. Available: http://doi.acm.org/10.1145/945445.945462

[16] S. U. R. Malik, S. U. Khan, and S. K. Srinivasan, "Modeling and analysis of state-of-the-art vm-based cloud management platforms," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, p. 1, 2013.

[17] G. Dasgupta, A. Sharma, A. Verma, A. Neogi, and R. Kothari, "Workload management for power efficiency in virtualized data centers," *Commun. ACM*, vol. 54, no. 7, pp. 131–141, Jul. 2011. [Online]. Available: http://doi.acm.org/10.1145/1965724.1965752

[18] North Bridge Venture Partner, "Future of Cloud Computing Survey," North Bridge Venture Partners, Tech. Rep., 2011.

[19] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, ser. MGC '10. New York, NY, USA: ACM, 2010, pp. 4:1–4:6. [Online]. Available: http://doi.acm.org/10.1145/1890799.1890803

[20] T. Setzer and A. Wolke, "Virtual machine re-assignment considering migration overhead," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, april 2012, pp. 631 –634.

[21] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang, "Energy-Aware Autonomic Resource Allocation in Multitier Virtualized Environments," *Services Computing, IEEE Transactions on*, vol. 5, no. 1, pp. 2 –19, jan.-march 2012.

[22] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or lets be friends," *Computer Networks*, vol. 53, no. 17, pp. 2905–2922, Dec. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2009.08.011

[23] S. Akoush, R. Sohan, A. Rice, A. W. Moore, and A. Hopper, "Predicting the Performance of Virtual Machine Migration," *Modeling, Analysis, and Simulation of Computer Systems, International Symposium on*, vol. 0, pp. 37–46, Aug. 2010. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5581607' escapeXml='false'/¿

[24] A. Gulati and A. Holler, "VMware Distributed Resource Management (DRS): Design, Implementation, and Lessons Learned," United States Environmental Protection Agency, Tech. Rep., 2012.

[25] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines," *Journal of Computer Networks*, vol. 53, no. 17, pp. 2923–2938, 2009. [Online]. Available: http://www.usenix.org/events/nsdi07/tech/wood.html http://linkinghub.elsevier.com/retrieve/pii/S1389128609002035

[26] A. Verma, P. Ahuja, and A. Neogi, "pmapper: Power and migration cost aware application placement in virtualized systems," in *Middleware 2008*, ser. Lecture Notes in Computer Science, V. Issarny and R. Schantz, Eds. Springer Berlin Heidelberg, 2008, vol. 5346, pp. 243–264.

[27] J. D. Hamilton, *Time Series Analysis*. Princeton University Press, 1994.

[28] W. Hu, A. Hicks, L. Zhang, E. M. Dow, V. Soni, H. Jiang, R. Bull, and J. N. Matthews, "A quantitative study of virtual machine live migration," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*. ACM, 2013, p. 11.

[29] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or let's be friends," *Comput. Netw.*, vol. 53, no. 17, pp. 2905–2922, 2009.

[30] A. Stage, "A Study of Resource Allocation Methods in Virtualized Enterprise Data Centres," TU Muenchen, Tech. Rep., 2013.

[31] S. Piramuthu, "On learning to predict Web traffic," *Decision Support Systems*, vol. 35, no. 2, pp. 213–229, May 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=782390.782393

[32] S. Casolari and M. Colajanni, "Short-term prediction models for server management in Internet-based contexts," *Decision Support Systems*, vol. 48, no. 1, pp. 212–223, Dec. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1651932.1652175


**Andreas Wolke** received his Dipl.-Inf.(FH) and MSc in computer sciences from the University of Applied Sciences in Augsburg. He is now a PhD student at the Technische Universität (TU) München. His research focuses on the resource allocation strategies in virtualized enterprise data centers and cloud computing environments.

**Martin Bichler** is a full professor at the Department of Informatics at the Technische Universität (TU) München. He received his PhD and his habilitation from the Vienna University of Economics and Business. Martin has worked as a research fellow at the University of California, Berkeley, and as a research staff member at the IBM T.J. Watson Research Center, New York.

**Thomas Setzer** received his Masters degree from the University of Karlsruhe, Germany, and his Ph.D. degree from the Technische Universität München, Germany. He is a professor at the Karlsruhe Institute of Technology (KIT) on analytics, dimensionality reduction and IT service management topics.